

Clustering Boolean Tensors

Saskia Metzler
Max-Planck-Institut für Informatik
Saarbrücken, Germany
smetzler@mpi-inf.mpg.de

Pauli Miettinen
Max-Planck-Institut für Informatik
Saarbrücken, Germany
pmiettin@mpi-inf.mpg.de

January 6, 2015

Abstract

Tensor factorizations are computationally hard problems, and in particular, are often significantly harder than their matrix counterparts. In case of Boolean tensor factorizations – where the input tensor and all the factors are required to be binary and we use Boolean algebra – much of that hardness comes from the possibility of overlapping components. Yet, in many applications we are perfectly happy to partition at least one of the modes. In this paper we investigate what consequences does this partitioning have on the computational complexity of the Boolean tensor factorizations and present a new algorithm for the resulting clustering problem. This algorithm can alternatively be seen as a particularly regularized clustering algorithm that can handle extremely high-dimensional observations. We analyse our algorithms with the goal of maximizing the similarity and argue that this is more meaningful than minimizing the dissimilarity. As a by-product we obtain a PTAS and an efficient 0.828-approximation algorithm for rank-1 binary factorizations. Our algorithm for Boolean tensor clustering achieves high scalability, high similarity, and good generalization to unseen data with both synthetic and real-world data sets.

1 Introduction

Tensors, multi-way generalizations of matrices, are becoming more common in data mining. Binary 3-way tensors, for example, can be used to record ternary relations (e.g. source IP—target IP—target port in network traffic analysis), a set of different relations between the same entities (e.g. subject—relation—object data for information extraction), or different instances of the same relation between the same entities (e.g. the adjacency matrices of a dynamic graph over time, such as who sent email to whom in which month). Given such data, a typical goal in data mining is to find some structure and regularities from it. To that end, tensor decomposition methods are often employed, and if the data is binary, it is natural to restrict also the factors to be binary. This gives rise to the various Boolean tensor decompositions that have recently been studied (e.g. Miettinen, 2011; Erdős and Miettinen, 2013b; Cerf et al, 2013).

Finding good Boolean tensor decompositions is, however, computationally hard. Broadly speaking, this hardness stems from the complexity caused by the overlapping factors. But when the existing Boolean tensor decomposition algorithms are applied to real-world data sets, they often return factors that are non-overlapping in at least one mode. Typically the mode where these non-overlapping factors appear is unsurprising given the data: in the above examples, it would be target ports in network data, relations in information extraction data, and the time (month) in the email data – in all of these cases, this mode has somewhat different behavior compared to the other two.

These observations lead to the question: what if we restrict one mode to non-overlapping factors? This removes one of the main sources of the computational complexity – the overlapping factors – from one mode, so we would expect the problem to admit better approximability results. We would also expect the resulting algorithms to be simpler. Would the computationally simpler problem transfer to better results? In this paper we study all these questions and, perhaps surprisingly, give an affirmative yes answer to all of them.

Normally, the quality of a clustering or tensor decomposition is measured using the distance of the original data to its representation. In Section 3.3 we argue, however, that for many data mining problems (especially with binary data), measuring the quality using the similarity instead of distance leads to a more meaningful analysis. Motivated by this, the goal of our algorithms is to maximize the similarity (Section 4.1) instead of minimizing the dissimilarity. As a by-product of the analysis of our algorithms, we also develop and analyse algorithms for maximum-similarity binary rank-1 decompositions in Section 4.2. We show that the problem admits a polynomial-time approximation scheme (PTAS) and present a scalable algorithm that achieves a $2(\sqrt{2} - 1) \approx 0.828$ approximation ratio. The experimental evaluation, in Section 5, shows that our algorithm achieves comparable representations of the input tensor when compared to methods that do (Boolean or normal) tensor CP decompositions – even when our algorithm is solving a more restricted problem – while being generally faster and admitting good generalization to unseen data.

2 Preliminaries

Throughout this paper, we indicate vectors as bold lower-case letters (\mathbf{v}), matrices as bold upper-case letters (\mathbf{M}), and tensors as bold upper-case calligraphic letters (\mathcal{T}). Element (i, j, k) of a 3-way tensor \mathcal{X} is denoted as x_{ijk} . A colon in a subscript denotes taking that mode entirely; for example, $\mathbf{X}_{::k}$ is the k th *frontal slice* of \mathcal{X} (\mathbf{X}_k in short).

A tensor can be *unfolded* into a matrix by arranging its fibers (i.e. its columns, rows, or tubes in case of a 3-way tensor) as columns of a matrix. For a *mode- n matricization*, mode- n fibers are used as the columns and the result is denoted by $\mathbf{X}_{(n)}$.

The outer product of vectors is denoted by \boxtimes . For vectors \mathbf{a} , \mathbf{b} , and \mathbf{c} of length n , m , and l , $\mathcal{X} = \mathbf{a} \boxtimes \mathbf{b} \boxtimes \mathbf{c}$ is an n -by- m -by- l tensor with $x_{ijk} = a_i b_j c_k$.

For a tensor \mathcal{X} , $|\mathcal{X}|$ denotes its number of non-zero elements. The Frobenius norm of a 3-way tensor \mathcal{X} is $\|\mathcal{X}\| = \sqrt{\sum_{i,j,k} x_{ijk}^2}$. If \mathcal{X} is binary, $|\mathcal{X}| = \|\mathcal{X}\|^2$. The *similarity* between two n -by- m -by- l binary tensors \mathcal{X} and \mathcal{Y} is defined as $\text{sim}(\mathcal{X}, \mathcal{Y}) = nml - |\mathcal{X} - \mathcal{Y}|$.

The *Boolean tensor sum* of binary tensors \mathcal{X} and \mathcal{Y} is defined as $(\mathcal{X} \vee \mathcal{Y})_{ijk} = x_{ijk} \vee y_{ijk}$. For binary matrices \mathbf{X} and \mathbf{Y} where \mathbf{X} has r columns and \mathbf{Y} has r rows their *Boolean matrix product*, $\mathbf{X} \circ \mathbf{Y}$, is defined as $(\mathbf{X} \circ \mathbf{Y})_{ij} = \bigvee_{k=1}^r x_{ik} y_{kj}$. The *Boolean matrix rank* of a binary matrix \mathbf{A} is the least r such that there exists a pair of binary matrices (\mathbf{X}, \mathbf{Y}) of inner dimension r with $\mathbf{A} = \mathbf{X} \circ \mathbf{Y}$.

A binary matrix \mathbf{X} is a *cluster assignment matrix* if each row of \mathbf{X} has exactly one non-zero element. In that case the Boolean matrix product corresponds to the regular matrix product,

$$\mathbf{X} \circ \mathbf{Y} = \mathbf{XY} . \quad (1)$$

We can now define the Boolean tensor rank and two decompositions: Boolean CP and Tucker3.

Definition 1. The *Boolean tensor rank* of a 3-way binary tensor \mathcal{X} , $\text{rank}_B(\mathcal{X})$, is the least integer r such that there exist r triplets of binary vectors $(\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i)$ with $\mathcal{X} = \bigvee_{i=1}^r \mathbf{a}_i \boxtimes \mathbf{b}_i \boxtimes \mathbf{c}_i$.

The low-rank Boolean tensor CP decomposition is defined as follows.

Problem 1 (Boolean CP). Given an n -by- m -by- l binary tensor \mathcal{X} and an integer r , find binary matrices \mathbf{A} (n -by- r), \mathbf{B} (m -by- r), and \mathbf{C} (l -by- r) such that they minimize $|\mathcal{X} - \bigvee_{i=1}^r \mathbf{a}_i \boxtimes \mathbf{b}_i \boxtimes \mathbf{c}_i|$.

The standard (non-Boolean) tensor rank and CP decomposition are defined analogously (Kolda and Bader, 2009). Both finding the least error Boolean CP decomposition and deciding the Boolean tensor rank are NP-hard (Miettinen, 2011). Following Kolda and Bader (2009), we use $[[\mathbf{A}, \mathbf{B}, \mathbf{C}]]$ to denote the normal 3-way CP decomposition and $[[\mathbf{A}, \mathbf{B}, \mathbf{C}]]_B$ for the Boolean CP decomposition.

Let \mathbf{X} be n_1 -by- m_1 and \mathbf{Y} be n_2 -by- m_2 matrix. Their *Kronecker (matrix) product*, $\mathbf{X} \otimes \mathbf{Y}$, is the $n_1 n_2$ -by- $m_1 m_2$ matrix defined by

$$\mathbf{X} \otimes \mathbf{Y} = \begin{pmatrix} x_{11}\mathbf{Y} & x_{12}\mathbf{Y} & \cdots & x_{1m_1}\mathbf{Y} \\ x_{21}\mathbf{Y} & x_{22}\mathbf{Y} & \cdots & x_{2m_1}\mathbf{Y} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n_1 1}\mathbf{Y} & x_{n_1 2}\mathbf{Y} & \cdots & x_{n_1 m_1}\mathbf{Y} \end{pmatrix}.$$

The *Khatri–Rao (matrix) product* of \mathbf{X} and \mathbf{Y} is defined as “column-wise Kronecker”. That is, \mathbf{X} and \mathbf{Y} must have the same number of columns ($m_1 = m_2 = m$), and their Khatri–Rao product $\mathbf{X} \odot \mathbf{Y}$ is the $n_1 n_2$ -by- m matrix defined as

$$\mathbf{X} \odot \mathbf{Y} = (\mathbf{x}_1 \otimes \mathbf{y}_1, \mathbf{x}_2 \otimes \mathbf{y}_2, \dots, \mathbf{x}_m \otimes \mathbf{y}_m). \quad (2)$$

Notice that if \mathbf{X} and \mathbf{Y} are binary, so are $\mathbf{X} \otimes \mathbf{Y}$ and $\mathbf{X} \odot \mathbf{Y}$.

We can write the Boolean CP as matrices using unfolding and matrix products:

$$\begin{aligned} \mathbf{X}_{(1)} &= \mathbf{A} \circ (\mathbf{C} \odot \mathbf{B})^T \\ \mathbf{X}_{(2)} &= \mathbf{B} \circ (\mathbf{C} \odot \mathbf{A})^T \\ \mathbf{X}_{(3)} &= \mathbf{C} \circ (\mathbf{B} \odot \mathbf{A})^T. \end{aligned} \quad (3)$$

The Boolean Tucker3 decomposition can be seen as a generalization of the Boolean CP decomposition:

Problem 2 (Boolean Tucker3). Given an n -by- m -by- l binary tensor \mathcal{X} and three integers r_1 , r_2 , and r_3 , find a binary r_1 -by- r_2 -by- r_3 core tensor \mathcal{G} and binary factor matrices \mathbf{A} (n -by- r_1), \mathbf{B} (m -by- r_2), and \mathbf{C} (l -by- r_3) such that they minimize

$$\left| \mathcal{X} - \bigvee_{\alpha=1}^{r_1} \bigvee_{\beta=1}^{r_2} \bigvee_{\gamma=1}^{r_3} g_{\alpha\beta\gamma} \mathbf{a}_\alpha \boxtimes \mathbf{b}_\beta \boxtimes \mathbf{c}_\gamma \right|. \quad (4)$$

We use $[[\mathcal{G}; \mathbf{A}, \mathbf{B}, \mathbf{C}]]_B$ as the shorthand notation for the Boolean Tucker3 decomposition.

3 Problem Definitions

We consider the variation of *tensor clustering* where the idea is to cluster one mode of a tensor and potentially reduce the dimensionality of the other modes. Another common approach is to do the co-clustering equivalent, that is, to cluster each mode of the tensor simultaneously. The former is the approach taken, for example, by Huang et al (2008), while the latter appears for instance in Jegelka et al (2009). Unlike either of these methods, however, we concentrate on binary data endowed with the Boolean algebra.

3.1 General Problem

Assuming a 3-way tensors and that we do the clustering in the last mode, we can express the general *Boolean tensor clustering* (BTC) problem as follows:

Problem 3 (BTC). Given a binary n -by- m -by- l tensor \mathcal{X} and integers k_1 , k_2 , and k_3 , find a k_1 -by- k_2 -by- k_3 binary tensor \mathcal{G} and matrices $\mathbf{A} \in \{0, 1\}^{n \times k_1}$, $\mathbf{B} \in \{0, 1\}^{m \times k_2}$, and $\mathbf{C} \in \{0, 1\}^{l \times k_3}$ such that \mathbf{C} is a cluster assignment matrix and that $[[\mathcal{G}; \mathbf{A}, \mathbf{B}, \mathbf{C}]]_B$ is a good Boolean Tucker decomposition of \mathcal{X} .

If we let $k_1 = n$ and $k_2 = m$, we obtain essentially a traditional (binary) clustering problem: Given a set of l binary matrices $\{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_l\}$ (the frontal slices of \mathcal{X}), cluster these matrices into k_3 clusters, each represented by an n -by- m binary matrix \mathbf{G}_i (the frontal slices of the core tensor \mathcal{G}); the factor matrices \mathbf{A} and \mathbf{B} can be left as identity matrices. We call this problem *Unconstrained BTC*. Writing each of the involved matrices as an nm -dimensional vector, we obtain the following problem, called the *Discrete Basis Partitioning problem* (DBPP) in (Miettinen et al, 2008):

Problem 4 (Miettinen et al DBPP, 2008). Given an l -by- nm binary matrix \mathbf{X} and a positive integer k , find matrices $\mathbf{C} \in \{0, 1\}^{l \times k}$ and $\mathbf{G} \in \{0, 1\}^{k \times nm}$ such that \mathbf{C} is a cluster assignment matrix and \mathbf{C} and \mathbf{G} minimize $\|\mathbf{X} - \mathbf{C}\mathbf{G}\|$.

Miettinen et al (2008) show that DBPP is NP-hard and give a $(10 + \epsilon)$ -approximation algorithm that runs in polynomial time w.r.t. n , m , l , and k , while Jiang (2014) gives a 2-approximation algorithm that runs in time $O(nml^k)$.

3.2 Boolean CP Clustering

Constraining k_1 , k_2 , and \mathcal{G} yields to somewhat different looking problems, though. In what can be seen as the other extreme, we can restrict $k_1 = k_2 = k_3 = k$ and \mathcal{G} (which now is k -by- k -by- k) to hyperdiagonal to obtain the *Boolean CP clustering* (BCPC) problem:

Problem 5 (BCPC). Given a binary n -by- m -by- l tensor \mathcal{X} and an integer k , find matrices $\mathbf{A} \in \{0, 1\}^{n \times k}$, $\mathbf{B} \in \{0, 1\}^{m \times k}$, and $\mathbf{C} \in \{0, 1\}^{l \times k}$ such that \mathbf{C} is a cluster assignment matrix and that $[[\mathbf{A}, \mathbf{B}, \mathbf{C}]]_B$ is a good Boolean CP decomposition of \mathcal{X} .

What BCPC does is perhaps easiest to understand if we use the unfolding rules (3) and write

$$\mathbf{X}_{(3)} \approx \mathbf{C} \circ (\mathbf{B} \odot \mathbf{A})^T, \quad (5)$$

where we can see that compared to the general BTC, we have restricted the type of centroids: each centroid must be a row of type $(\mathbf{b} \otimes \mathbf{a})^T$. This restriction plays a crucial role in the decomposition, as we shall see shortly. Notice also that using (1) we can rewrite (5) without the Boolean product:

$$\mathbf{X}_{(3)} \approx \mathbf{C}(\mathbf{B} \odot \mathbf{A})^T. \quad (6)$$

3.3 Similarity vs. Dissimilarity

So far we have avoided defining what we mean by “good” clustering. Arguably the most common approach is to say that any clustering is good if it minimizes the sum of distances (i.e. dissimilarities) between the original elements and their cluster’s representative (or centroid). An alternative is to maximize the similarity between the elements and their representatives: if the data and the centroids are binary, this problem is known as *Hypercube segmentation* (Kleinberg et al, 2004). While maximizing similarity is obviously a dual of minimizing the dissimilarity – in the sense that an optimal solution to one is also an optimal solution to the other – the two behave differently when we aim at approximating the optimal solution. For example, for a fixed k , the best known approximation algorithm to DBPP is the aforementioned 2-approximation algorithm (Jiang, 2014), while Alon and Sudakov (1999) gave a PTAS for the Hypercube segmentation problem.

The differences between the approximability, and in particular the reasons behind those differences, are important for data miners. Namely, it is not uncommon that our ability to minimize the dissimilarities is bounded by a family of problems where the optimal solution obtains extremely small dissimilarities, while the best polynomial-time algorithm has to do with slightly larger errors. These larger errors, however, might still be small enough relative to the data size that we can ignore them. This is what essentially happens when we maximize the similarities.

For example, consider an instance of DBPP where the optimal solution causes ten errors. The best-known algorithm can guarantee to not cause more than twenty errors; assuming the data size is, say, 1000-by-500, making mistakes on twenty elements can still be considered excellent. On the other hand, should the optimal solution make an error on every fourth element, the best approximation could only guarantee to make errors in at most every second element. When considering the similarities instead of dissimilarities, these two setups look very different: In the first, we get within 0.99998 of the best solution, while in the other, we are only within 0.66667 of the optimal. Conversely, if the algorithm wants to be within, say, 0.9 of the optimal similarity in the latter setup, it must approximate the dissimilarity within a factor of 1.3. Hence similarity is

Algorithm 1 SaBoTeur algorithm for the BCPC

Input: 3-way binary tensor \mathcal{X} , number of clusters k , number of samples r .

Output: Binary factor matrices \mathbf{A} and \mathbf{B} , cluster assignment matrix \mathbf{C} .

```
1: function SaBoTeur( $\mathcal{X}, k, r$ )
2:   repeat
3:     Sample  $k$  rows of  $\mathbf{X}_{(3)}$  into matrix  $\mathbf{Y}$ 
4:     Find binary matrices  $\mathbf{A}$  and  $\mathbf{B}$  that maximize  $\text{sim}(\mathbf{Y}, (\mathbf{B} \odot \mathbf{A})^T)$ 
5:     Cluster  $\mathbf{C}$  by assigning each row of  $\mathbf{X}_{(3)}$  to its closest row of  $(\mathbf{B} \odot \mathbf{A})^T$ 
6:   until  $r$  resamples are done
7:   return best  $\mathbf{A}$ ,  $\mathbf{B}$ , and  $\mathbf{C}$ 
```

less sensitive to small errors (relative to the data size) and more sensitive to large errors, than dissimilarity is. In many applications, this is exactly what we want.

None of this is to say that we should stop considering the error and concentrate only on the similarities. On the contrary: knowing the data size and the error, we can easily compute the similarity for the binary data, and especially when the error is relatively small, it is much easier to compare than the similarity scores. What we do argue, however, is that when analyzing the approximation ratio a data mining algorithm can obtain, similarity can give us a more interesting picture of the actual behavior of the algorithm. Hence, for the rest of the paper, we shall concentrate on the maximum-similarity variants of the problems; most notably, we concentrate on the maximum-similarity BCPC, denoted BCPC_{\max} .

4 Solving the BCPC_{\max}

Given a tensor \mathcal{X} , for the optimal solution to BCPC_{\max} , we need matrices \mathbf{A} , \mathbf{B} , and \mathbf{C} that maximize $\text{sim}(\mathbf{X}_{(3)}, \mathbf{C}(\mathbf{B} \odot \mathbf{A})^T)$. If we replace $\mathbf{B} \odot \mathbf{A}$ with an arbitrary binary matrix, this would be equal to the Hypercube segmentation problem defined by Kleinberg et al (2004): Given a set S of l vertices of the d -dimensional cube $\{0, 1\}^d$, find k vertices $P_1, \dots, P_k \in \{0, 1\}^d$ and a partition of S into k segments to maximize $\sum_{i=1}^k \sum_{c \in S} \text{sim}(P_i, c)$. Hence our algorithm resembles those for Hypercube segmentation, with the added restrictions to the centroid vectors.

4.1 The Algorithm

Alon and Sudakov (1999) gave an algorithm for the Hypercube segmentation problem that obtains a similarity within $(1 - \varepsilon)$ of the optimum. The running time of the algorithm is $e^{O((k^2/\varepsilon^2) \ln k)} nml$ for n -by- m -by- l data. While technically linear in data size, the first term turns the running time unfeasible even for moderate values of k (the number of clusters) and ε . We therefore base our algorithm on the simpler algorithm by Kleinberg et al (2004) that is based on random sampling. This algorithm obtains an approximation ratio of $0.828 - \varepsilon$ with constant probability and running time $O(nmlk(9/\varepsilon)^k \ln(1/\varepsilon))$. While the running time is still exponential in k , it is dominated by the number of samples we do: each sample takes time $O(nmlk)$ for k clusters and n -by- m -by- l data. For practical purposes, we can keep the number of samples constant (with the cost of losing approximation guarantees, though).

Our algorithm **SaBoTeur** (Sampling for Boolean Tensor clustering), Algorithm 1, considers only the unfolded tensor $\mathbf{X}_{(3)}$. In each iteration, it samples k rows of $\mathbf{X}_{(3)}$ as the initial, unrestricted centroids. It then turns these unrestricted centroids into the restricted type in line 4, and then assigns each row of $\mathbf{X}_{(3)}$ to its closest restricted centroid. The sampling is repeated multiple times, and in the end, the factors that gave the highest similarity are returned.

The algorithm is extremely simple and fast. In line 3 the algorithm samples k rows of the data as its initial centroids. Kleinberg et al (2004) proved that among the rows of $\mathbf{X}_{(3)}$ that are in the same optimal cluster, one is a good approximation of the (unrestricted) centroid of the cluster:

Lemma 1 (Kleinberg et al 2004, Lemma 3.1). *Let \mathbf{X} be an n -by- m binary matrix and let*

$$\mathbf{y}^* = \arg \max_{\mathbf{y} \in \{0,1\}^m} \sum_{i=1}^n \text{sim}(\mathbf{x}_i, \mathbf{y}) .$$

Then there exist a row \mathbf{x}_j of \mathbf{X} such that

$$\sum_{i=1}^n \text{sim}(\mathbf{x}_i, \mathbf{x}_j) \geq (2\sqrt{2} - 2) \sum_{i=1}^n \text{sim}(\mathbf{x}_i, \mathbf{y}^*) . \quad (7)$$

That is, if we sample one row from each cluster, we have a high probability of inducing a close-optimal clustering.

4.2 Binary Rank-1 Matrix Decompositions

The next part of the **SaBoTeur** algorithm is to turn the unrestricted centroids into the restricted form (line 4). We start by showing that this problem is equivalent to finding the maximum-similarity binary rank-1 decomposition of a binary matrix:

Problem 6 (Binary rank-1 decomposition). Given an n -by- m binary matrix \mathbf{X} , find an n -dimensional binary vector \mathbf{a} and an m -dimensional binary vector \mathbf{b} that maximize $\text{sim}(\mathbf{X}, \mathbf{a} \boxtimes \mathbf{b})$.

Lemma 2. *Given an k -by- nm binary matrix \mathbf{X} , finding n -by- k and m -by- k binary matrices \mathbf{A} , \mathbf{B} that maximize $\text{sim}(\mathbf{X}, (\mathbf{B} \odot \mathbf{A})^T)$ is equivalent to finding the most similar binary rank-1 approximation of each row \mathbf{x} of \mathbf{X} , where the rows are re-shaped as n -by- m binary matrices.*

Proof. If \mathbf{x}_i is the i th row of \mathbf{X} and \mathbf{z}_i is the corresponding row of $(\mathbf{B} \odot \mathbf{A})^T$, then $\text{sim}(\mathbf{X}, (\mathbf{B} \odot \mathbf{A})^T) = \sum_{i=1}^k \text{sim}(\mathbf{x}_i, \mathbf{z}_i)$, and hence we can solve the problem row-by-row. Let $\mathbf{x} = (x_{1,1}, x_{2,1}, \dots, x_{n,1}, x_{1,2}, \dots, x_{n,m})$ be a row of \mathbf{X} . Re-write \mathbf{x} as an n -by- m matrix \mathbf{Y} ,

$$\mathbf{Y} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix} .$$

Consider the row of $(\mathbf{B} \odot \mathbf{A})^T$ that corresponds to \mathbf{x} , and notice that it can be written as $(\mathbf{b} \otimes \mathbf{a})^T$, where \mathbf{a} and \mathbf{b} are the columns of \mathbf{A} and \mathbf{B} that correspond to \mathbf{x} . As $(\mathbf{b} \otimes \mathbf{a})^T = (b_1 \mathbf{a}^T, b_2 \mathbf{a}^T, \dots, b_m \mathbf{a}^T)$, re-writing it similarly as \mathbf{x} we obtain

$$\begin{pmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_m \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_m \\ \vdots & \vdots & \ddots & \vdots \\ a_n b_1 & a_n b_2 & \cdots & a_n b_m \end{pmatrix} = \mathbf{a} \mathbf{b}^T = \mathbf{a} \boxtimes \mathbf{b} .$$

Therefore, $\text{sim}(\mathbf{x}, (\mathbf{b} \otimes \mathbf{a})^T) = \text{sim}(\mathbf{Y}, \mathbf{a} \boxtimes \mathbf{b})$. □

We start by showing that the maximum-similarity binary rank-1 decomposition admits a PTAS. To that end, we present a family of randomized algorithms that on expectation attain a similarity within $(1 - \varepsilon)$ of the optimum. The family is similar to that of Alon and Sudakov's (1999) and can be analysed and de-randomized following the techniques presented by them. The family of algorithms (one for each ε) is presented as Algorithm 2.

Lines 7 and 8 require us to solve one of the factor vectors when the other is given. To build \mathbf{b} (line 7), we simply take the column-wise majority element of those rows where \mathbf{a} is 1, and we extend \mathbf{a} similarly in line 8.

Algorithm 2 Randomized PTAS for maximum-similarity binary rank-1 decompositions

Input: An n -by- m binary matrix \mathbf{X} .

Output: Binary vectors \mathbf{a} and \mathbf{b} .

```
1: function PTAS $_{\varepsilon}(\mathbf{X})$ 
2:   Sample  $l = \Theta(\varepsilon^{-2})$  rows of  $\mathbf{X}$  u.a.r. with replacements
3:   for all partitions of the sample into two sets do
4:     Let  $\mathbf{X}'$  contain the rows in the sample
5:     for both sets in the partition do
6:       Let  $\mathbf{a}$  be the incidence vector of the set
7:       Find  $\mathbf{b}^T$  maximizing  $\text{sim}(\mathbf{X}', \mathbf{a}\mathbf{b}^T)$ 
8:       Extend  $\mathbf{a}$  to the rows not in the sample maximizing  $\text{sim}(\mathbf{X}, \mathbf{a}\mathbf{b}^T)$ 
9:   return best vectors  $\mathbf{a}$  and  $\mathbf{b}$ 
```

Algorithm 3 Approximation of maximum-similarity binary rank-1 decompositions

Input: An n -by- m binary matrix \mathbf{X} .

Output: Binary vectors \mathbf{a} and \mathbf{b} .

```
1: function A( $\mathbf{X}$ )
2:   for all rows  $\mathbf{x}_i$  of  $\mathbf{X}$  do
3:     Let  $\mathbf{b} = \mathbf{x}_i$ 
4:     Find  $\mathbf{a}$  maximizing  $\text{sim}(\mathbf{X}, \mathbf{a}\mathbf{b}^T)$ 
5:   return best vectors  $\mathbf{a}$  and  $\mathbf{b}$ 
```

The analysis of Algorithm 2 follows closely the proofs by Alon and Sudakov (1999) and is omitted.

The running time of the PTAS algorithm becomes prohibitive even with moderate values of ε , and therefore we will not use it in **SaBoTeur**. Instead, we present a simple, deterministic algorithm that approximates the maximum similarity within 0.828, Algorithm 3. It is similar to the algorithm for Hypercube segmentation based on random sampling presented by Kleinberg et al (2004). The algorithm considers every row of \mathbf{X} as a potential vector \mathbf{b} and finds the best \mathbf{a} given \mathbf{b} . Using Lemma 1 it is straight forward to show that the algorithm achieves the claimed approximation ratio:

Lemma 3. *Algorithm 3 approximates the optimum similarity within 0.828 in time $O(nm \min\{n, m\})$.*

Proof. To prove the approximation ratio, let $\mathbf{a}^*(\mathbf{b}^*)^T$ be the optimum decomposition. Consider the rows in which \mathbf{a}^* has 1. Per Lemma 1, selecting one of these rows, call it \mathbf{b} , gives us $\text{sim}(\mathbf{X}, \mathbf{a}^*\mathbf{b}^T) \geq (2\sqrt{2} - 2)\text{sim}(\mathbf{X}, \mathbf{a}^*\mathbf{b}^T)$ (notice that $\mathbf{a}^*\mathbf{b}^T$ agrees with the optimal solution in rows where \mathbf{a}^* is zero). Selecting \mathbf{a} that maximizes the similarity given \mathbf{b} can only improve the result, and the claim follows as we try every row of \mathbf{X} .

If $n < m$, the time complexity follows as for every candidate \mathbf{b} we have to make one sweep over the matrix. If $m < n$, we can operate on the transpose. \square

4.3 Iterative Updates

The **SaBoTeur** algorithm bears resemblance to the initialization phase of k -means style clustering algorithms. We propose a variation of **SaBoTeur**, **SaBoTeur+itUp**, where k -means style iterative updates are performed on the result of **SaBoTeur** until it does not improve further. In each iteration of the iterative updates phase, the new centroids are first determined by placing 1s in those positions where the majority of cluster members have a 1. Next, these new centroids are constrained to rank-1, and finally, the cluster assignment is recomputed.

This procedure clearly slows down the algorithm. Also, due to the constraint on the centroids the iterative updates might actually impair the result, unlike in the normal k -means algorithm that converges to a local optimum. However, as we shall see in Section 5, the results in practice improve slightly.

4.4 Implementation Details

We implemented the **SaBoTeur** algorithm in C.¹ In the implementation, the tensor \mathcal{X} is represented as a bitmap. This allows for using fast vectorized instructions such as `xor` and `popcnt`. In addition this representation takes exactly one bit per entry (excluding necessary padding), being very memory efficient even compared to sparse tensor representations.

The second ingredient to the fast algorithm we present is parallelization. As every row of $\mathbf{X}_{(3)}$ is handled independently, the algorithm is embarrassingly parallel. We use the OpenMP (Dagum and Menon, 1998) and parallelize along the sampled cluster centroids. This means that the rank-1 decompositions of each centroid as well as the computation of the similarity in each of the columns of the Khatri-Rao product (line 4 of Algorithm 1) are computed in parallel.

4.5 Selecting the Number of Clusters

A common problem in data analysis is that many methods require the selection of the *model order* a priori; for example, most low-rank matrix factorization methods require the user to provide the target rank before the algorithm starts. Many clustering methods – ours included – assume the number of clusters as a parameter, although there are other methods that do not have this requirement.

When the user does not know what would be a proper number of clusters for the data, we propose the use of the *minimum description length principle* (MDL, Rissanen, 1978) for automatically inferring it. In particular, we adopt the recent work of Miettinen and Vreeken (2014) on using MDL for Boolean matrix factorization (BMF) to our setting (we refer the reader to Miettinen and Vreeken (2014) for more information).

The intuition behind the MDL principle is that the best model for the data (best number of clusters, in our case) is the one that lets us compress the data most. We use the two-part (i.e. crude) MDL where the *description length* contains two parts: the description length of the model M , $L(M)$, and that of the data D given the model M , $L(D | M)$. The overall description length is simply $L(M) + L(D | M)$. The optimal model (in MDL’s sense) is the one where $L(M) + L(D | M)$ is minimized.

In BCPC, the model consists of the two factor matrices \mathbf{A} and \mathbf{B} and the cluster assignment matrix \mathbf{C} . Given these, we can reconstruct the original tensor \mathcal{X} if we know the locations where $[[\mathbf{A}, \mathbf{B}, \mathbf{C}]]_B$ differs from \mathcal{X} . Therefore, $L(M)$ is the total length of encoding \mathbf{A} , \mathbf{B} , and \mathbf{C} , while $L(D | M)$ is the length of encoding the locations of the differences, i.e. tensor $\mathcal{X} \oplus [[\mathbf{A}, \mathbf{B}, \mathbf{C}]]_B$, where \oplus is the element-wise *exclusive-or*.

Encoding \mathbf{A} and \mathbf{B} is similar to BMF, so we can use the DtM encoding of Miettinen and Vreeken (2014) for encoding them (and the related dimensions of the data). To encode the matrix \mathbf{C} , we only need to store to which cluster each frontal slice is associated to, taking $l \cdot \log_2(k)$ bits. This finalises the computation of $L(M)$.

To compute $L(D | M)$, we note that we can re-write $\mathcal{X} \oplus [[\mathbf{A}, \mathbf{B}, \mathbf{C}]]_B = \mathbf{X}_{(3)} \oplus \mathbf{C} \circ (\mathbf{B} \circ \mathbf{A})^T$. The computation of $L(D | M)$ now becomes equivalent of computing it for Boolean matrix factorization with factor matrices \mathbf{C} and $\mathbf{D} = (\mathbf{B} \circ \mathbf{A})^T$. Hence, we can follow the Typed XOR DtM approach of Miettinen and Vreeken (2014) directly.

To sum up, in order to use MDL to select the number of clusters, we have to run **SaBoTeur** with different numbers of clusters, compute the description length for each result, and take the one that yields the smallest description length.

4.6 Maximum-Similarity BTC

We note here that as the Hypercube segmentation problem is the maximization version of the DBPP (Problem 4), the algorithms of Kleinberg et al (2004) and Alon and Sudakov (1999) can be used as such for solving the maximum-similarity Unconstrained BTC.

¹The code is available from <http://www.mpi-inf.mpg.de/~pmiettinen/btc/>

5 Experimental Evaluation

5.1 Other Methods and Evaluation Criteria

To the best of our knowledge, this paper presents the first algorithms for Boolean tensor clustering and hence we cannot compare directly to other methods. We decided to compare **SaBoTeur** to continuous and Boolean tensor CP decompositions. We did not use other tensor clustering methods as they aim at optimizing significantly different targets (see Section 6 for more elaborate discussion).

We used the **BCP-ALS** (Miettinen, 2011) and **Walk’n’Merge** (Erdős and Miettinen, 2013b) algorithms for computing Boolean CP decompositions. **BCP-ALS** is based on iteratively updating the factor matrices one at a time (similarly to the classical alternating least squares optimizations), while **Walk’n’Merge** is a recent algorithm for scalable Boolean tensor factorization in sparse binary tensors. We did not use **Walk’n’Merge** on synthetic data as **BCP-ALS** is expected to perform better on smaller and denser tensors (Erdős and Miettinen, 2013b), but we used it on some larger real-world tensors; **BCP-ALS**, on the other hand, does not scale well to larger tensors and hence we had to omit it from most real-world experiments.

Of the continuous methods we used **ParCube** (Papalexakis et al, 2012)² and **CP-APR** (Chi and Kolda, 2012) (implementation from the Matlab Tensor Toolbox v2.5³). **CP-APR** is an alternating Poisson regression algorithm that is specifically developed for sparse (counting) data (which can be expected to follow the Poisson distribution), with the goal of returning sparse factors.

The other method, **ParCube**, uses sampling to find smaller sub-tensors. It then solves the CP decomposition in this sub-tensor, and merges the solutions back into one. We used a non-negative variant of **ParCube** that expects non-negative data, and returns non-negative factor matrices.

For synthetic data, we report the relative similarity, that is, the fraction of the elements where the data and the clustering agree. For real-world data, we report the error measured using squared Frobenius norm (i.e. the number of disagreements between the data and the clustering when both are binary). Comparing binary methods against continuous ones causes issues, however. Using the squared Frobenius can help the real-valued methods, as it scales all errors less than 1 down, but at the same time, small errors cumulate unlike with fully binary data. To alleviate this problem, we also rounded the reconstructed tensors from **CP-APR** and **ParCube** to binary tensors. We tried different rounding thresholds between 0 and 1 and selected the one that gave the lowest (Boolean) reconstruction error. The rounded versions are denoted by **CP-APR**_{0/1} and **ParCube**_{0/1}.

It is worth emphasizing that all of the methods we are comparing against are solving a relaxed version of the BCPC problem. Compared to BCPC, the Boolean CP factorization is not restricted to clustering the third mode while the normal CP factorization lifts even the requirement for binary factor matrices. Hence, a valid solution to BCPC is always also a valid solution to (Boolean and normal) CP factorization (notice, however, that the solutions for Boolean and normal CP factorization are not interchangeable). The methods we compare against should therefore always perform better than (or at least as good as) **SaBoTeur**.

5.2 Synthetic Experiments

To test the **SaBoTeur** algorithm in a controlled environment, we created synthetic data sets that measured the algorithm’s response to (1) different numbers of clusters, (2) different density of data, (3) different levels of additive noise, and (4) different levels of destructive noise. All tensors were of size 700-by-500-by-50. All data sets were created by first creating ground-truth binary factor matrices **A**, **B**, and **C**. The default number of clusters was 7 and the default density of the tensor $\mathcal{X} = [[\mathbf{A}, \mathbf{B}, \mathbf{C}]]_B$ was 0.05.

Additive and destructive noise were applied to the tensor \mathcal{X} . Additive noise turns zeros into ones while destructive noise turns ones into zeros. The default noise level⁴ for both types was 10%, yielding to a noised input tensor $\tilde{\mathcal{X}}$.

²<http://www.cs.cmu.edu/~epapalex/>

³<http://www.sandia.gov/~tgkolda/TensorToolbox/>

⁴The noise levels are reported w.r.t. number of non-zeros.

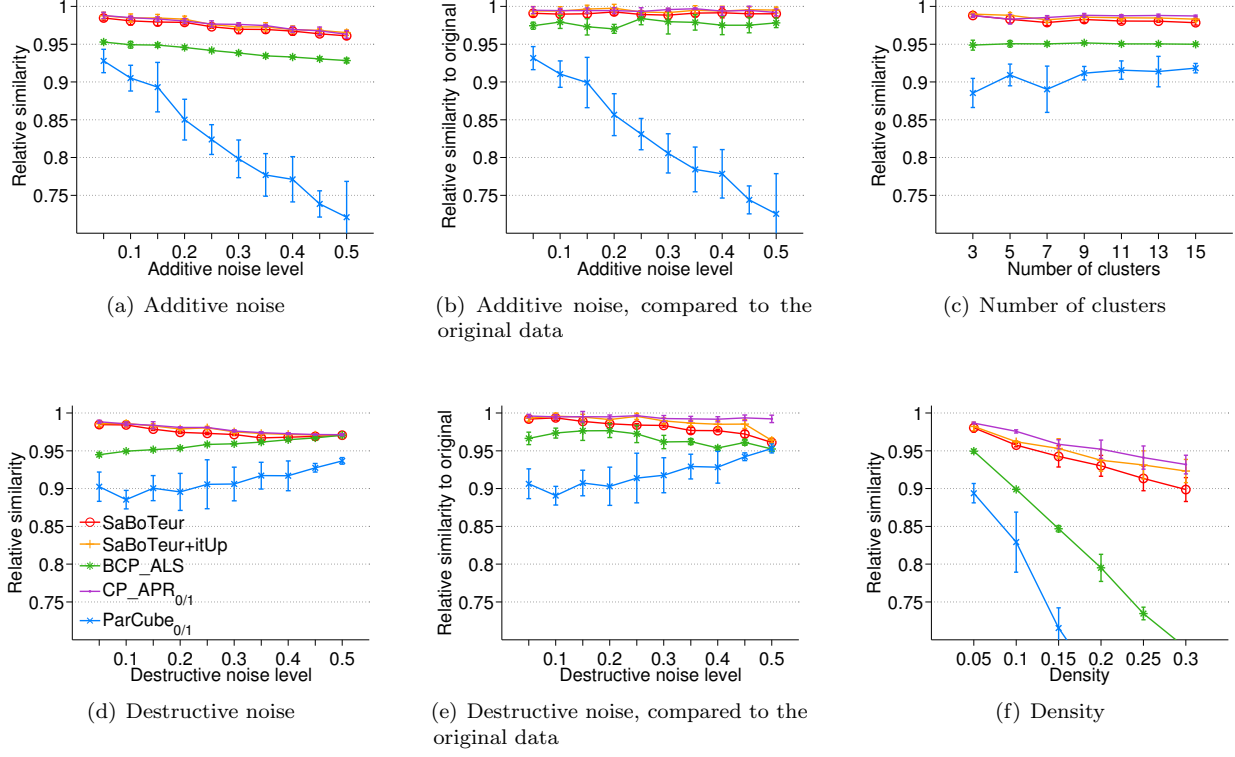


Figure 1: Results from synthetic experiments when varying different data characteristics. All y -axes show the relative similarity. All markers are mean values over 5 iterations and the width of the error bars is twice the standard deviation.

We varied each of the four features one at a time keeping the others in their default values, and created 5 random copies on each parameter combination. The results we report are mean values over these five random copies. In all experiments, the number of clusters (or factors) was set to the true number of clusters used to create the data. The number of re-samples in **SaBoTeur** was set to $r = 20$ in all experiments.

For similarity experiments, we compared the obtained reconstructions $[[\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}]]_B$ to both the original tensor \mathcal{X} and the noised tensor $\tilde{\mathcal{X}}$.

5.2.1 Varying the Noise

In the first experiment, we studied the effects different noise levels have to the reconstruction accuracy. First, we varied the level of additive noise from 5% to 50% (in steps of 5%). The results are in Figure 1(a), where we can see that **SaBoTeur** (with and without iterative updates) and **CP_APR_{0/1}** all achieve very high similarity with a gentle downwards slope as the level of additive noise increases. **BCP_ALS** is slightly worse, but consistent, while the performance of **ParCube_{0/1}** suffers significantly from increased noise levels. In order to test if the good performance of **SaBoTeur** means it is just modeling the noise, we also compared the similarity of the reconstruction to the original, noise-free tensor \mathcal{X} (Figure 1(b)). This did not change the results in meaningful ways, except that **BCP_ALS** improved somewhat.

When moving from additive to destructive noise (Figure 1(d)), **SaBoTeur**, **SaBoTeur+itUp**, and **CP_APR_{0/1}** still stay as the best three methods, but as the destructive noise level increases, **BCP_ALS** approaches the three. That behaviour, however, is mostly driven by ‘modeling the noise’, as can be seen in Figure 1(e), which shows the similarity to the noise-free data. There, with highest levels of destructive noise, **SaBoTeur**’s performance suffers more than **CP_APR_{0/1}**’s.

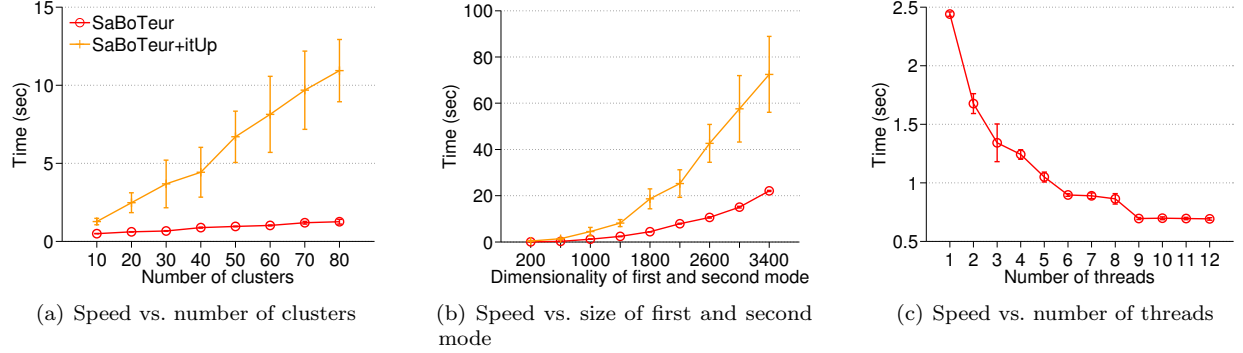


Figure 2: Results from scalability tests. All times are wall-clock times. Markers are mean values over 5 iterations and the width of the error bars is twice the standard deviation.

5.2.2 Varying the Number of Clusters

The number of clusters varied from 3 to 15 with steps of 2. The results are shown in Figure 1(c). None of the tested methods show any significant effect to the number of clusters, and the best three methods are still SaBoTeur, SaBoTeur+itUp, and CP-APR_{0/1}.

5.2.3 Varying the Density

The density of the tensor varied from 5% to 30% with steps of 5%. The results can be seen in Figure 1(f). All methods perform worse with denser data, with CP-APR_{0/1}, SaBoTeur+itUp, and SaBoTeur being the best three methods in that order. BCP-ALS and ParCube’s results quickly went below 75% similarity, ParCube_{0/1} going as low as 55% with 30% density (we omit the worst results from the plots for clarity). Note that an increased density implies higher amounts of noise as the level of noise is defined with respect to the number of ones.

5.2.4 Scalability

We run the scalability tests on a dedicated machine with two Intel Xeon X5650 6-Core CPUs at 2.66GHz and 64GB of main memory. All reported times are wall-clock times. For these experiments, we created a new set of tensors with a default size of 800-by-800-by-500, a density of 5%, 10% of additive and destructive noise, and 20 clusters by default. During the experiments, we varied the size, the number of clusters, and the number of threads used for the computation.

We also tried ParCube, CP-APR, and BCP-ALS with these data sets, but only ParCube was able to finish any of the data sets. However, already with 200-by-200-by-500 data, the smallest we tried, it took 155.4 seconds. With 10 clusters, the least number we used, ParCube finished only after 1319.4 seconds. Therefore, we omitted these results from the figures.

As can be seen in Figure 2(a), the SaBoTeur algorithm scales very well with the number of clusters. This is mostly due to efficient parallelization of the computing of the clusters (c.f. below). SaBoTeur+itUp, however, slows down with higher number of clusters. This is to be expected, given that more clusters require more update computations.

For the second experiment, we varied the dimensionality of the first and second mode between 200 and 3400 with steps of 400. The results can be seen in Figure 2(b). As we grew both modes simultaneously, the size of the tensor grows as a square (and, as the density was kept constant, so does the number of non-zeros). Given this, the running time of SaBoTeur grows as expected, or even slower, while SaBoTeur+itUp is again clearly slower.

In the last scalability experiment (Figure 2(c)), we tested how well SaBoTeur parallelizes. As the computer

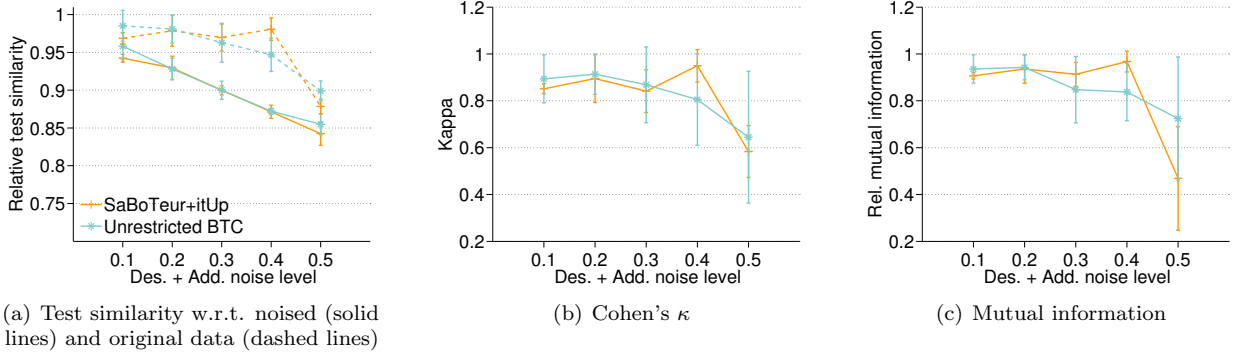


Figure 3: Evaluation of the generalization quality using (a) reconstruction error on test data, and (b) Kappa statistic and (c) mutual information of cluster labels between the obtained and true labels in the test data. Markers are mean values over 5 different data sets and the width of the error bars is twice the standard deviation.

used had 12 cores, we set that as the maximum number of threads. Yet, after 9 threads, there were no more obvious benefits. We expect that the memory bus is the limiting factor here.

5.2.5 Generalization Tests

In our final test with synthetic data, we tested how well **SaBoTeur+itUp**'s clusterings generalize to yet-unseen data. Our hypothesis is that restricting the centroids to rank-1 matrices helps with the overfitting, and hence we compared **SaBoTeur+itUp** to unrestricted Boolean tensor clustering (BTC), where the centroids are arbitrary binary matrices. Notice that Unrestricted BTC will always obtain at least as good reconstruction error on the training data as **SaBoTeur+itUp**.

We generated tensors of size 700-by-500-by-300 with 7 clusters, 15% of density and different levels of both additive and destructive noise, from 10% till 50%. We randomly selected 25% of the frontal slices as the test data and computed the clusterings on the remaining data. We then connected each frontal slice in the test set to its closest centroid.

The quality of the results was measured in two different ways. First, we used the reconstruction error in the test data (Figure 3(a)) and computed the overall similarity of expressing the testing data using the centroids. We also used the original (i.e. noise-free) frontal slices to assess whether we 'model the noise'. The results show that with the noise-free test data (dashed lines), **SaBoTeur+itUp** is better than the unrestricted BTC with 30–40% noise, and the results are reversed with 10% and 50% of noise. With noisy test data, the differences are less.

We also measured the similarity of the cluster labels w.r.t. the ground truth. We used two metrics for that, Cohen's κ statistic (Figure 3(b)) and normalized mutual information (Figure 3(c)). Cohen's κ takes values from $[-1, 1]$, -1 meaning complete disagreement and 1 meaning complete agreement. As Cohen's κ requires the clusters' labels to match, we paired the labels using bipartite maximum matching (we used the Hungarian method (Papadimitriou and Steiglitz, 1998) to compute the matching). The mutual information was normalized by dividing it with the joint entropy; the resulting statistic takes values from $[0, 1]$, with 1 meaning perfect agreement.

The two measures agree that **SaBoTeur+itUp** is better at 40% of noise and worse with 10% and 50% of noise. With 10% of noise, the difference is not significant with either metric, and with 50% of noise, the difference is significant only with normalized mutual information.

Overall, the results show that **SaBoTeur+itUp** has a small advantage over the unrestricted BTC. Perhaps the clearest sign of the benefits of the rank-1 centroids is the consistently smaller standard deviation **SaBoTeur+itUp** obtains. This suggests that **SaBoTeur+itUp** is less susceptible to random variations in the

Table 1: Size and density of the real-world datasets.

Dataset	Rows	Columns	Tubes	Density (10^{-7})
Delicious	1640	23584	7968	8.23
Enron	146	146	38	22752.86
Facebook	42390	39986	224	16.51
Last.FM	1892	12523	9749	8.07
MovieLens	2113	5908	9079	4.23
MAG	10197	10673	20	1036.37
Resolver	343	360	200	626.01
TracePort	10266	8622	501	2.51
YAGO	190962	62428	25	67.35

data, which should improve the generalization quality. Indeed, our generalization experiments with the real-world data (Section 5.3.5) support this.

5.3 Real-World Data

We tested **SaBoTeur** also with multiple real-world data sets of varying characteristics. The purpose of these experiments is to verify our findings with synthetic data. To that end, we studied how well **SaBoTeur** (and the other methods) can reconstruct the data, how the methods scale with real-world data, and how **SaBoTeur** generalizes to unknown data. In addition, we also studied the sparsity of the factors, using MDL to select the number of clusters, and how interpretable the results **SaBoTeur** provides are.

5.3.1 Data Sets

We used nine real-world data sets. The size and density for each data set is listed in Table 1. The Delicious data⁵ (Cantador et al, 2011) contains user–bookmark–tag tuples from the Delicious social bookmarking system⁶. The Enron data⁷ contains information about who sent an e-mail to whom (rows and columns) per months (tubes). The Facebook data set⁸ (Viswanath et al, 2009) contains information about who posted a message on whose wall per week. The Last.FM data⁵ (Cantador et al, 2011) consists of artist–user–tag tuples from the Last.FM online music system⁹. The MovieLens data set and the MAG data set are obtained from the same source⁵ (Cantador et al, 2011). While MovieLens is composed of user–movie–tag tuples, MAG consists of movies–actors–genres tuples. The Resolver data contains entity–relation–entity tuples from the TextRunner open information extraction algorithm¹⁰ (Yates and Etzioni, 2009). We used the sample called ResolverL in Miettinen (2011). The TracePort data set¹¹ contains anonymized passive traffic traces (source and destination IP and port numbers) from 2009. The YAGO data set consist of subject–object–relation tuples from the semantic knowledge base YAGO¹² (Suchanek et al, 2007).

As a preprocessing step we removed all the all-zero slices in every mode. In addition, for the Delicious data set, also all slices that had less than 6 entries were purged. For the MAG data, all actors that appeared in less than 5 movies were discarded.

⁵<http://grouplens.org/datasets/hetrec-2011>

⁶<http://www.delicious.com>

⁷<http://www.cs.cmu.edu/~enron/>

⁸<http://socialnetworks.mpi-sws.org/datasets.html>

⁹<http://www.last.fm>

¹⁰<http://www.cis.temple.edu/~yates/papers/jair-resolver.html>

¹¹http://www.caida.org/data/passive/passive_2009_dataset.xml

¹²<http://www.mpi-inf.mpg.de/yago-naga/yago>

5.3.2 Reconstruction Accuracy

As all real-world data sets are very sparse, we report the errors, not the similarity, for these experiments. The error is measured as the number of disagreements, when the reconstructed tensor is also binary, or as squared tensor Frobenius distance, when the reconstructed tensor is real-valued (not-rounded versions of **CP-APR** and **ParCube**). The results can be seen in Table 2. The results for **CP-APR**_{0/1} and **ParCube**_{0/1} for all data sets except **Enron** and **Resolver** are obtained by sampling: If $|\mathcal{X}|$ is the number of non-zeros in data \mathcal{X} , we sampled $200|\mathcal{X}|$ locations of 0s, and computed the error the methods make for every non-zero and for the sampled zero elements. The sampled results were then extrapolated to the size of the full tensor. We had to use sampling, as the factor matrices were too dense to allow reconstructing the full tensor. We will discuss more on density below.

The smallest reconstruction error is obtained by **ParCube**_{0/1} in almost all experiments, **YAGO** being a notable exception. Remember, however, that **ParCube**_{0/1} returns a non-negative tensor CP decomposition that is afterwards rounded to binary tensor, that is, it is neither clustering nor Boolean, and hence together with **CP-APR**_{0/1} is expected to be better than **SaBoTeur**. Also, without the rounding, **ParCube** is often the worst method by a large margin. **CP-APR** benefits much less from the rounding, **CP-APR**_{0/1} being comparable to **SaBoTeur**.

In those data sets where we were able to run **BCP-ALS**, its results were consistently worse than **SaBoTeur**’s results, despite it solving more relaxed problem. We were unable to get **Walk’n’Merge** to finish within a reasonable time with most data sets: it found rank-1 tensors sufficiently quickly, but took too much time to select the top ones from there (see Erdős and Miettinen (2013b) for explanation on how **Walk’n’Merge** finds a Boolean CP decomposition). When it did finish, however, it was slightly better than **SaBoTeur+itUp** or **SaBoTeur**.

Finally, the iterative updates of **SaBoTeur+itUp** consistently improved the results compared to **SaBoTeur**, but did so with significant increase to the running time. It seems that with most data sets, the iterative updates are not necessarily worth the extra wait.

5.3.3 Sparsity of the Factors

An important question on the practical feasibility of the tensor decomposition and clustering algorithms is the density of the factor matrices. Too dense factor matrices increase the computational complexity and also the storage requirements. Furthermore, multiplying the dense factors together becomes prohibitively expensive, making it impossible to fully re-construct the tensor. This is the reason why, for example, we had to use sampling to compute the rounded representations of **CP-APR** and **ParCube**.

We studied the sparsity of the factors using the **Delicious** data. In Table 3 we report the total densities of the matrices **A** and **B**, that is, $(|\mathbf{A}| + |\mathbf{B}|)/k(n + m)$ for n -by- k and m -by- k factors.

It is obvious that the Boolean methods, **SaBoTeur** and **Walk’n’Merge**, are orders of magnitude sparser than the continuous methods (**CP-APR** and **ParCube**). This was to be expected as similar behaviour has already been observed with Boolean matrix factorizations (Miettinen, 2010).

We did not compare the third-mode factor matrices for densities. For **SaBoTeur**, the clustering assignment matrix **C** has density $1/l$ that depends only on the number of frontal slices l ; obtaining density less than that requires having rows of **C** that have no non-zeros.

5.3.4 Scalability with Real-World Data

Table 4 shows the wall-clock times for **MovieLens** and **Resolver** with $k = 15$. **MovieLens** is one of the sparsest data sets, while **Resolver** is one of the densest ones, and hence these two show us how the relative speed of the methods changes as the density changes. The other data sets generally followed the behaviour we report here, adjusting to their density.

With the **MovieLens** data, **ParCube** is the fastest of the methods, followed by **SaBoTeur**. **CP-APR**, **Walk’n’Merge**, and **SaBoTeur+itUp** are significantly slower than the two. With the denser **Resolver** data, however, the order is changed, with **SaBoTeur** and **SaBoTeur+itUp** being an order of magnitude faster than **Walk’n’Merge**,

Table 2: Reconstruction errors on real data sets, measured as squared tensor Frobenius distance. ‘—’ means that the algorithm was not able to finish.

	$k =$	Delicious				Enron				Last.FM				MAG		
		7	10	15	30	5	10	12	15	8	15	20	30	5	7	
SaBoTeur	253608	253546	253447	253004	1811	1779	1769	1753	195905	195735	195570	195278	224322	223580		
SaBoTeur+itUp	253153	253293	252738	252859	1793	1756	1750	1735	186072	185713	185651	185399	223869	223489		
BCP_ALS	—	—	—	—	1850	1850	1850	1850	—	—	—	—	—	—		
CP_APR	253558	253516	253433	253196	1718	1631	1598	1560	184513	184038	183489	182659	224996	224696		
CP_APR _{0/1}	253653	253653	253652	253639	1838	1817	1811	1781	180766	177527	174491	170939	225580	225155		
ParCube	367521	373710	363581	418252	2137	2273	2352	2270	2745259	2879381	2856941	2770361	375290	322121		
ParCube _{0/1}	247129	246566	246016	241850	1802	1744	1751	1690	133057	139207	119461	131322	222133	221933		
Walk'n'Merge	—	251034	—	—	—	—	—	1753	—	—	—	—	—	—		

	$k =$	MovieLens				Resolver				TracePort				Facebook			YAGO		
		5	15	20	30	5	10	15	30	5	10	15	30	15	20	7			
SaBoTeur	56761	56712	56678	56571	1530	1509	1485	1455	11085	11004	10923	10776	626999	626776	1957737				
SaBoTeur+itUp	47933	47479	47280	47316	1522	1503	1457	1443	10990	10961	10913	10673	—	—	—				
BCP_ALS	—	—	—	—	1624	1624	1626	1632	—	—	—	—	—	—	—				
CP_APR	47584	46927	46555	46287	1519	1509	1489	1457	11140	11114	11082	11027	626343	626202	—				
CP_APR _{0/1}	47178	44655	43614	42609	1545	1545	1538	1539	11110	11059	10893	10617	626945	626945	—				
ParCube	252719	463025	454826	449534	1784	1762	1798	1939	19803	30793	31107	29796	750891	797534	619476763				
ParCube _{0/1}	41223	38098	37915	37393	1507	1471	1450	1370	10708	9913	10097	9647	619939	620894	4793390				
Walk'n'Merge	—	46807	—	—	—	—	—	1534	—	—	10679	—	—	—	—				

Table 3: Total density of the factor matrices \mathbf{A} and \mathbf{B} for different k on the Delicious data.

$k =$	7	10	15	30
SaBoTeur	0.00016	0.00031	0.00029	0.00029
CP_APR	0.20219	0.14786	0.10312	0.05396
ParCube	0.28186	0.21069	0.16658	0.09900
Walk'n'Merge	—	0.00022	—	—

Table 4: Average wall-clock running times in seconds for real-world data sets. Averages are over 5 re-starts for MovieLens and 10 for Resolver. Both data sets used $k = 15$ clusters.

	MovieLens	Resolver
SaBoTeur	45.6	0.02
SaBoTeur+itUp	303.6	0.11
CP_APR	196.6	22.00
ParCube	24.8	7.68
Walk'n'Merge	207.5	2.15

which still is faster than ParCube or CP_APR. The density of the data does not affect the speed of SaBoTeur, while it does have a significant effect to CP_APR and ParCube. SaBoTeur+itUp is less affected by density, and more affected by the size of the data, than the other methods.

5.3.5 Generalization with Real-World Data

We repeated the generalization test we did with synthetic data with three real-world data sets – MovieLens, Resolver, and TracePort – keeping 85% of the frontal slices as training data while the remaining 15% we used for testing. The results can be seen in Table 5. We used $k = 15$ clusters for MovieLens and TracePort, and 10 clusters for the Resolver data. On each data set we conducted 10 repetitions of the training phase, and selected the one with the smallest training error.

As expected, the Unrestricted BTC achieves lower training error, but larger testing error. This agrees with our synthetic experiments, and strongly indicates that using the rank-1 centroids helps to avoid overfitting.

5.3.6 Selecting the Number of Clusters

We tested the use of MDL to select the number of clusters with two real-world data sets, Enron and MAG. To that end, we ran SaBoTeur with every possible number of clusters (i.e. from having all slices in one cluster to having one cluster for each slice). Further, we computed the description length of representing the data with no clusters, i.e. having the full data explained in the $L(D | M)$ part. The description lengths for the different numbers of clusters are presented in Figure 4. For Enron, the number of clusters that gave the smallest description length was 3, while for MAG it was 16.

Table 5: Training and testing errors on real-world data. The number of clusters was $k = 15$ for MovieLens and TracePort and $k = 10$ for Resolver.

		MovieLens	Resolver	TracePort
Train	SaBoTeur+itUp	40075	1282	9730
	Unrestricted BTC	39351	1122	8843
Test	SaBoTeur+itUp	7258	225	1144
	Unrestricted BTC	8256	274	1554

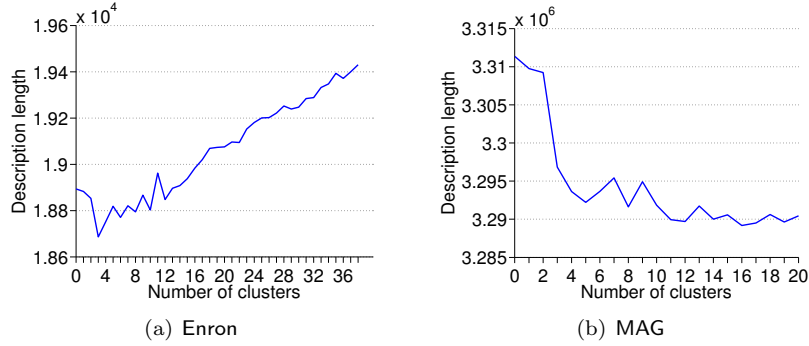


Figure 4: Description length vs. number of clusters.

Both Enron and MAG show non-smooth behaviour in the description length. This is likely partially due to SaBoTeur’s randomized behaviour and partially due to the fact that SaBoTeur does not consider the description length when it builds the clustering. Nevertheless, especially with Enron (Figure 4(a)), the selection of the best number of clusters – in the MDL’s sense – seems clear.

5.3.7 Interpretability of the Results

For the final experiment with the real-world data, we studied some of the results we obtained from SaBoTeur in order to see if they can provide insights to the data. We report results from three data sets: MAG, Delicious, and Last.FM. For MAG, we can interpret all three modes (movies, actors, and genres); for the other two, we can only interpret two of the three modes (tags and URLs, and tags and artists, respectively), as the third mode corresponds to user IDs.

For, MAG, we used 16 clusters (as suggested by MDL), and clustered the genres. As there are only 20 genres in the data, this means many clusters contained only one genre; this, however, is not surprising, as we would expect the sets of movies that fall in different genres be generally rather different. The results picked famous movies of the genre as the centroids: for example, the cluster of *animation* and *comedy* genres had the 1995 animation *Toy Story* appearing in its centroid. Another cluster, containing genres *adventure* and *fantasy*, had as its centroid Peter Jackson’s *Lord of the Rings* trilogy, and its main cast. Similarly to *Toy Story*, *Lord of the Rings* movies are arguably stereotypical adventure–fantasy movies.

The centroids for the MAG data contained very few movies (e.g. the three *Lord of the Rings* movies). This is due to the fact that the data sets are very sparse, and the algorithm can only pick up the most prominent elements in the centroids. With even sparse data sets, such as Delicious and Last.FM, the centroids get even sparse, perhaps containing just a singleton element from some mode. This is a common problem to many Boolean methods (see, e.g. Miettinen et al, 2008). To alleviate it, we used a method proposed by Miettinen et al (2008), and weighted the reconstruction error. Normally, representing a 0 as a 1 and representing a 1 as a 0 both increase the error by 1, but for the next experiments, we set the weight so that representing a 1 with a 0 causes 10 times more error as representing 0 as a 1.

For Delicious and Last.FM, we mined 30 clusters. Even with the 10-fold weight, results for Delicious were rather sparse (indicating that the users’ behaviour was very heterogeneous). Some clusters were very informative, however, such as one with tags *software*, *apple*, *mac*, and *osx* that contained bookmarked pages on *HFS for Windows*, *iVPN*, *Object-Oriented Programming with Objective-C*, and *iBooks and ePub*, among others. For Last.FM, the clusters were generally more dense, as users tend to tag artists more homogeneously. For example, the cluster containing the tags *pop* and *rnb* contained the superstar artists such as Beyoncé, Britney Spears, Katy Perry, and Miley Cyrus (and many more). But importantly, one could also find less-known artists: in a cluster containing tags *dance*, *trance*, *house*, *progressive trance*, *techno*, and *vocal trance*, we found artists such as *Infected Mushroom*, *T.M.Revolution*, *Dance Nation*, and *RuPaul*.

5.4 Discussion

All the experiments show that **SaBoTeur** can hold its own even against methods that are theoretically superior: For reconstruction error (or similarity) with both synthetic (Sections 5.2.1–5.2.3) and real-world data (Section 5.3.2), **SaBoTeur** (and **SaBoTeur+itUp**) achieve results that are the best or close to the best results, notwithstanding that other methods, especially **CP-APR_{0/1}** and **ParCube_{0/1}**, benefit from significantly less constrained forms of decomposition, coupled with the post-hoc rounding. These relaxations also come with a price, as both methods create significantly denser factor matrices, and – as can be seen in Sections 5.2.4 and 5.3.4 – are unable to scale to denser data. Compared to the Boolean CP factorization methods, **BCP-ALS** and **Walk’n’Merge**, **SaBoTeur** is comparable (or sometimes clearly better) in reconstruction error and density of the factor matrices, while being significantly more scalable.

Our hypothesis that the rank-1 centroids help with overfitting was also confirmed (Sections 5.2.5 and 5.3.5): While the Unrestricted BTC obtained lower training errors (as expected), **SaBoTeur** obtained better results with testing data in all real-world data sets and in most synthetic experiments (with **TracePort**, **SaBoTeur**’s testing error was more than 25% better than that of Unrestricted BTC’s).

Our experiments also show that MDL can be used to select the number of clusters automatically, even though we do think that more studies are needed to better understand the behaviour of the description length. Also, when studying the results, it seems obvious that **SaBoTeur** can return results that are easy to interpret and insightful; in part, this is due to the Boolean algebra, and in part, due to the clustering.

The last open question is whether one should use **SaBoTeur** or **SaBoTeur+itUp**. Judging by the results, **SaBoTeur+itUp** can provide tangible benefits over **SaBoTeur**, however, with a significant price in the running time. Thence, we think that it is best to start with **SaBoTeur**, and only if the user thinks she would benefit from more accurate results, move to **SaBoTeur+itUp**.

6 Related Work

Tensor clusterings have received some research interest in recent years. The existing work can be divided roughly into two separate approaches: on one hand Jegelka et al (2009) study the problem of clustering simultaneously all modes of a tensor (tensor co-clustering), and on the other hand, Huang et al (2008) and Liu et al (2010) (among others) study the problem where only one mode is clustered and the remaining modes are represented using a low-rank approximation. The latter form is closer to what we study in this paper, but the techniques used for the continuous methods do not apply to the binary case.

Normal tensor factorizations are well-studied, dating back to the late Twenties. The Tucker and CP decompositions were proposed in the Sixties (Tucker, 1966) and Seventies (Carroll and Chang, 1970; Harshman, 1970), respectively. The topic has nevertheless attained growing interest in recent years, both in numerical linear algebra and computer science communities. For a comprehensive study of recent work, see Kolda and Bader (2009), and the recent work on scalable factorizations by Papalexakis et al (2012).

The first algorithm for Boolean CP factorization was presented by Leenen et al (1999), although without much analysis. Working in the framework of formal tri-concepts, Bělohlávek et al (2012) studied the exact Boolean tensor decompositions, while Ignatov et al (2011) studied the problem of finding dense rank-1 subtensors from binary tensors. In data mining Miettinen (2011), studied computational complexity and sparsity of the Boolean tensor decompositions. Recently Erdős and Miettinen (2013b) proposed a scalable algorithm for Boolean CP and Tucker decompositions, and applied that algorithm for information extraction (Erdős and Miettinen, 2013a). The tri-concepts are also related to closed n -ary relations, that is, n -ary extensions of closed itemsets (see, e.g. Cerf et al, 2009, 2013). For more on these methods and their relation to Boolean CP factorization, see Miettinen (2011).

Miettinen et al (2008) presented the Discrete Basis Partitioning problem for clustering binary data and using binary centroids. They gave a $(10 + \varepsilon)$ approximation algorithm based on the idea that any algorithm that can solve the so-called binary graph clustering (where centroids must be rows of the original data) within factor f , can solve the arbitrary binary centroid version within factor $2f$. Recently, Jiang (2014) gave a 2-approximation algorithm for the slightly more general case with $k + 1$ clusters with one centroid restricted

to be an all-zero vector.

The maximization dual of binary clustering, the hypercube segmentation problem, was studied by Kleinberg et al (1998, 2004) and they also gave three algorithms for the problem, obtaining an approximation ratio of $2(\sqrt{2} - 1)$. Later, Seppänen (2005) proved that this ratio is tight for this type of algorithms and Alon and Sudakov (1999) presented a PTAS for the problem.

Recently, Kim and Candan (2011) proposed the tensor-relational model, where relational information is represented as tensors. They defined the relational algebra over the (decompositions of) the tensors, noting that in the tensor-relational model, the tensor decomposition becomes the costliest operation. Hence, their subsequent work (Kim and Candan, 2012, 2014) has concentrated on faster decompositions in the tensor-relational framework.

7 Conclusions and Future Work

We have studied the problem of clustering one mode of a 3-way binary tensor while simultaneously reducing the dimensionality in the two other modes. This problem bears close resemblance to the Boolean CP tensor decomposition, but the additional clustering constraint makes the problem significantly different. The main source of computational complexity, the consideration of overlapping factors in the tensor decomposition, does not play a role in BCPC. This lets us design algorithms with provable approximation guarantees better than what is known for the Boolean matrix and tensor decompositions.

Our experiments show that the **SaBoTeur** algorithm obtains comparable reconstruction errors compared to the much less restricted tensor factorization methods while obtaining sparse factor matrices and overall best scalability. We also showed that restricting the cluster centroids to rank-1 matrices significantly helped the generalization to unseen data.

The idea of restricting the clustering centroids to a specific structure could be applied to other types of data, as well. Whether that will help with the curse of dimensionality remains an interesting topic for the future research, but based on the results we presented in this paper, we think this approach could have potential. On the other hand, one could also consider higher-rank centroids, moving from BCPC towards Unrestricted BTC. In a sense, the rank of the centroid can be seen as a regularization parameter, but the viability of this approach requires more research, both in Boolean and in continuous domains.

We also argue that the similarity – as opposed to distance or error – is often more meaningful metric for studying the (theoretical) performance of data mining algorithms due to its robustness towards small errors and more pronounced effects of large errors. To assess whether similarity is better than dissimilarity, more methods, existing and novel, should be analysed, and the results should be contrasted to those obtained when studying the dissimilarity and real-world performance. Our hypothesis is that good theoretical performance in terms of similarity correlates better with good real-world performance than good (or bad) performance correlates with the error.

Finally, our algorithms could be used to provide fast decompositions of Boolean tensors in the tensor-relational model.

References

- Alon N, Sudakov B (1999) On two segmentation problems. *J Algorithm* 33:173–184
- Bělohlávek R, Glodeanu C, Vychodil V (2012) Optimal Factorization of Three-Way Binary Data Using Triadic Concepts. *Order* 30(2):437–454
- Cantador I, Brusilovsky P, Kuflik T (2011) 2nd Workshop on Information Heterogeneity and Fusion in Recommender Systems. In: *RecSys '11*
- Carroll JD, Chang JJ (1970) Analysis of individual differences in multidimensional scaling via an N-way generalization of “Eckart–Young” decomposition. *Psychometrika* 35(3):283–319

- Cerf L, Besson J, Robardet C, Boulicaut JF (2009) Closed patterns meet n-ary relations. *ACM Trans Knowl Discov Data* 3(1)
- Cerf L, Besson J, Nguyen KNT, Boulicaut JF (2013) Closed and noise-tolerant patterns in n-ary relations. *Data Min Knowl Discov* 26(3):574–619
- Chi EC, Kolda TG (2012) On Tensors, Sparsity, and Nonnegative Factorizations. *SIAM J Matrix Anal Appl* 33(4):1272–1299
- Dagum L, Menon R (1998) OpenMP: an industry standard API for shared-memory programming. *IEEE Comput Sci Eng Mag* 5(1):46–55
- Erdős D, Miettinen P (2013a) Discovering Facts with Boolean Tensor Tucker Decomposition. In: *CIKM '13*, pp 1569–1572
- Erdős D, Miettinen P (2013b) Walk'n'Merge: A Scalable Algorithm for Boolean Tensor Factorization. In: *ICDM '13*, pp 1037–1042
- Harshman RA (1970) Foundations of the PARAFAC procedure: Models and conditions for an “explanatory” multimodal factor analysis. Tech. Rep. 16, UCLA Working Papers in Phonetics
- Huang H, Ding C, Luo D, Li T (2008) Simultaneous tensor subspace selection and clustering: The equivalence of high order SVD and k-means clustering. In: *KDD '08*, pp 327–335
- Ignatov DI, Kuznetsov SO, Magizov RA, Zhukov LE (2011) From Triconcepts to Triclusters. In: *RSFDGrC '11*, pp 257–264
- Jegelka S, Sra S, Banerjee A (2009) Approximation Algorithms for Tensor Clustering. In: *ALT '09*, pp 368–383
- Jiang P (2014) Pattern extraction and clustering for high-dimensional discrete data . PhD thesis, University of Illinois at Urbana-Champaign
- Kim M, Candan KS (2011) Approximate tensor decomposition within a tensor-relational algebraic framework. In: *CIKM '11*, pp 1737–1742
- Kim M, Candan KS (2012) Decomposition-by-normalization (DBN): Leveraging approximate functional dependencies for efficient tensor decomposition. In: *CIKM '12*, pp 355–364
- Kim M, Candan KS (2014) Pushing-down tensor decompositions over unions to promote reuse of materialized decompositions. In: *ECML PKDD '14*, pp 688–704
- Kleinberg J, Papadimitriou C, Raghavan P (1998) A Microeconomic View of Data Mining. *Data Min Knowl Discov* 2(4):311–324
- Kleinberg JM, Papadimitriou CH, Raghavan P (2004) Segmentation problems. *J ACM* 51(2):263–280
- Kolda TG, Bader BW (2009) Tensor decompositions and applications. *SIAM Rev* 51(3):455–500
- Leenen I, Van Mechelen I, De Boeck P, Rosenberg S (1999) INDCLAS: A three-way hierarchical classes model. *Psychometrika* 64(1):9–24
- Liu X, De Lathauwer L, Janssens F, De Moor B (2010) Hybrid Clustering of Multiple Information Sources via HOSVD. In: *ISNN '10*, pp 337–345
- Miettinen P (2010) Sparse Boolean Matrix Factorizations. In: *ICDM '10*, pp 935–940
- Miettinen P (2011) Boolean Tensor Factorizations. In: *ICDM '11*, pp 447–456

- Miettinen P, Vreeken J (2014) MDL4BMF: Minimum description length for Boolean matrix factorization. *ACM Trans Knowl Discov Data* 8(4)
- Miettinen P, Mielikäinen T, Gionis A, Das G, Mannila H (2008) The Discrete Basis Problem. *IEEE Trans Knowl Data En* 20(10):1348–1362
- Papadimitriou CH, Steiglitz K (1998) *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineola, New York
- Papalexakis EE, Faloutsos C, Sidiropoulos ND (2012) ParCube: Sparse Parallelizable Tensor Decompositions. In: *ECML PKDD '12*, pp 521–536
- Rissanen J (1978) Modeling by shortest data description. *Automatica* 14(5):465–471
- Seppänen JK (2005) Upper bound for the approximation ratio of a class of hypercube segmentation algorithms. *Inform Process Lett* 93(3):139–141
- Suchanek FM, Kasneci G, Weikum G (2007) Yago: A core of semantic knowledge. In: *WWW '07*, pp 697–706
- Tucker LR (1966) Some mathematical notes on three-mode factor analysis. *Psychometrika* 31(3):279–311
- Viswanath B, Mislove A, Cha M, Gummadi KP (2009) On the evolution of user interaction in Facebook. In: *WOSN '09*, pp 37–42
- Yates A, Etzioni O (2009) Unsupervised methods for determining object and relation synonyms on the web. *J Artif Intell Res* 34:255–296